Journal of Electronic Testing (2023) 39:387–402
https://doi.org/10.1007/s10836-023-06067-6

Incomplete Testing of SOC

Kunwer Mrityunjay Singh[1] · Jatindra Deka[1] · Santosh Biswas[2]

Received: 28 January 2023 / Accepted: 27 April 2023 / Published online: 29 May 2023

Abstract
Nowadays, System on Chip (SOC) based devices such as smartphones, tablets, cameras, and others are commonly used. The cost of these devices is determined by the expenses associated with their manufacturing and testing. In modern manufacturing technology, SOC-based devices have more cores embedded within them. However, testing these numerous cores thoroughly can be quite expensive and can sometimes cost more than the manufacturing itself. To make these devices more affordable for people from economically weaker backgrounds, it is necessary to come up with an efficient testing strategy that can help reduce the costs. In this study, we introduce a new method of testing the SOC incompletely instead of testing it thoroughly. This method involves compromising on the test quality, which may result in errors in the output. Incomplete testing is performed only for those cores of the SOC that can tolerate such errors. For example, incomplete testing is performed for the cores that are responsible for multimedia applications such as image or video display, where a slight compromise in the quality can be tolerated by human eyes. This incomplete testing helps to reduce the *Test Power Consumption* (TP), *Test Access Tim*e (TAT), and *Test Data Volume* (TDV) while compromising with the *Fault Coverage* (FC).

Keywords SOC · Power aware testing · Test data volume · Test access time · Fault coverage · TAM

1 Introduction

The SOC is widely utilized in various products, including smart phones, tablets, and digital cameras. These products undergo rigorous testing to meet customers' demands for affordability and versatility. Due to SOCs, digital devices now have the capability to perform a wide range of tasks, such as paying bills, booking tickets, and taking photos and videos. In order to afford these devices, some low-income customers may compromise on certain aspects, such as accepting lower screen resolutions or sacrificing multimedia applications. Additionally, the fast-paced nature of today's world means that new models are frequently introduced, leading customers to replace their devices frequently. Consequently, most customers do not require devices that are durable for long periods; rather, they prioritize economic devices. Testing plays a critical role in achieving cost-effectiveness. As the number of applications in a product increases, more cores are added to the SOC, which can affect other factors such as TDV, TAT, and TP. An efficient testing method can help reduce the overall cost of a product.

To recede TDV, TAT and TP various compression techniques are proposed. The most popular methods for reducing TDV are compression and compaction. In various research papers, compression techniques have been proposed, such as Golomb codes-based test vector compression and decompression mechanism in [5, 9], and [7] to encode precomputed test vectors for cores in the SOC. Run-length encoding-based compression for test resource partitioning has been proposed in [8] to reduce TDV, TP, and TAT. The use of Huffman coding techniques for compression has also been discussed in [20] and [27]. Compression-based techniques aim to minimize the area overhead introduced by coder and decoder circuits. Similarly, the selective

Responsible Editor: K. Chakrabarty

Jatindra Deka and Santosh Biswas contributed equally to this work.

✉ Kunwer Mrityunjay Singh
mrityunjay.jkit@gmail.com

Jatindra Deka
jatin@iitg.ac.in

Santosh Biswas
santoshbiswas402@yahoo.com

[1] Department of Computer Science, IIT, Guwahati 781039, Assam, India

[2] Department of Electrical Engineering and Computer Science, IIT, Bhilai 492015, Chhattisgarh, India

Huffman coding and complementary Huffman coding-based compression method has been discussed in [41]. Compaction techniques, such as test vector reordering and don't care bit filling, have also been found effective for reducing TDV. For instance, in [25], test vector reordering has been used to increase fault coverage and minimize the number of test vectors needed to cover all the faults. Don't care bit filling has also been found effective for reducing TDV, as shown in [30] and [29], where X compactor has been introduced. Some methods have also used compression and compaction techniques simultaneously to reduce TDV, such as run-length coding and hamming-based test vector reordering implemented in [28]. In [2], a hybrid technique has been introduced, which uses a block-matching algorithm to rearrange test vector blocks in subsequent test patterns to separate low-frequency and high-frequency data sets of test vectors. Other compression and compaction techniques have also been proposed in [24, 40], and [1]. All of these well-known compression and compaction techniques greatly reduce test data volume, but require a decompresser on a chip which adds area overhead. With the increasing size of SOC, the size of TDV also increases, resulting in a higher probability of bit flipping and power consumption. Several methods have been proposed to reduce TP, such as assigning binary values to don't care bits and compressing test data using Golomb code [6], filling don't care bits and using run length coding compression [36], predicting the impact of filling don't care bits on capture power [23], and enhancing the correlation between successive test vectors to reduce power profiles [33]. The large size of test vector sets due to the increasing number of cores in SOC makes storing and transporting test vector data a challenge, which impacts TAT and TP. As a result, storage of TDV, area overhead, software program storage, large TAT, and large TP are the main testing challenges nowadays. Several approaches have been utilized to overcome challenges posed by large TDV, TAT, and TP. These approaches have employed heuristic techniques, such as those put forth in [11, 17], and [16]. A potential solution using Genetic algorithms (GA) is presented in [11] for managing test scheduling and test access mechanism partition in SOCs. In [17], a GA-based strategy is proposed for jointly optimizing test scheduling and wrapper design while adhering to power constraints for core-based SOCs. Finally, method in [16] suggests a GA-based methodology for modular testing of hierarchical SOCs that comprise of earlier generation SOCs as embedded mega cores. To improve TDV, TAT and TP, designers have used various methods, including approximate computing techniques. These techniques relax the requirement for exact computation and instead aim for acceptable levels of accuracy. This results in significant improvements in TAT and TP performance without a significant increase in cost [26]. For example, the Approx-Multiply-Accumulate(MAC) unit is an approximate circuit proposed in [13] that utilizes the concept of approximate computing. It uses an approximate hybrid redundant adder to perform internal multiplication and addition operations, which reduces hardware size and TP with a slight tradeoff in computation error. In another method, a MAC arithmetic architecture, introduced in [32], reduces the number of intermediate partial components generated during arithmetic operations by a factor of 2, resulting in improved speed and reduced hardware cost. These methods, as explained in [13] and [32], are based on the idea that certain systems can tolerate output errors to some extent, and exploiting this can lead to significant benefits in terms of reduced hardware area and TP. Conventional methods to test digital circuits are given in [39]. Another approximate aware testing method is proposed for approximate circuits in [10], which involves a fault classification algorithm that categorizes faults into two groups: approximation-redundant faults and non-approximation faults. This method has some limitations because *Automatic Test Pattern Generation*(ATPG) tools prioritize the shortest propagation paths of faults to the primary outputs, which may result in incorrect classification of non-approximation faults that propagate to multiple primary outputs as approximation-redundant faults. To address this limitation, another ATPG methodology for approximate circuits is introduced in [15]. This methodology is based on boolean satisfiability and takes into account the quality of the output as well as the difference between non-approximation faults and approximation-redundant faults. The utilization of approximate testing is beneficial in various fault-tolerant applications such as audio, video, graphics, and wireless communications, as mentioned in [37]. In [12], a case study is presented where a mobile device receives streaming H.264 video over a WCDMA wireless channel. The video compression algorithms take advantage of the temporal correlation of image sequences and assume that adjacent pictures have small differences, allowing many parts of the current frame to be borrowed from previously decoded frames stored in the decoded picture buffer (DPB). However, the DPB requires a large memory space and can dominate the SOC design of H.264. To reduce TP, H.264 uses spatial redundancy and allows some errors to occur at the chip level. Another example is a wireless communication application discussed in [14]. In this application, data is received and digitized, then presented to the baseband digital modem, which estimates the transmitted stream and presents it to the decoding section. Channel decoders correct errors and generate an error-free stream depending on the application. Redundancy is inherent to data stream communication in wireless communication, and data buffering memories within an SOC must be error-tolerant as the data stored in them will eventually be processed by the channel decoder. These data buffering memories constitute the majority of the memory in wireless applications. Multi-Level Cells

STT-RAM [35], neural networks [38], and convolutional neural networks [31] are some of the other applications suitable for approximate testing at both the circuit and system levels. The utilization of approximate circuits can accelerate computation and enhance testing metrics. However, designing such circuits is a demanding, intricate, and time-consuming task that necessitates a thorough comprehension of the circuit. Furthermore, testing these circuits is complex because specialized techniques are required to verify that the outcomes meet the requirements of the intended application.

In this paper, we propose a testing method that tests the SOC incompletely and tolerates errors in output without modifying the circuit's behavior, unlike approximate circuits. This testing method demonstrates a significant improvement in testing parameters, such as TDV, TAT, and TP, with only a minor compromise in the quality of testing. Section 2 describes the problem statement, while section 3 outlines the proposed method. Sections 4, 5, and 6 demonstrate the impact of incomplete testing on FC, TDV, TAT, and TP. Section 8 presents the experimental results, section 9 provides the conclusion, and section 10 explores future work.

2 Problem Formulation

Given a SOC, generate reduced test vectors set for incomplete testing of SOC to reduced TDV, TAT, TP and maximize the fault coverage.

3 Proposed Method

To understand our proposed method we first focus on Fig. 1 which is nothing but the multiplication of two 3-bit binary numbers. Figure 1 displays two 3-bit binary numbers, A2, A1, A0 and B2, B1, B0, where A0 and B0 are the *Least Significant Bits* (LSBs). These numbers are multiplied to produce six bits, P5 to P0, with P0 being the LSB and P4 the *Most Significant Bit* (MSB). Each bit in the output has a weight. Figure 1 illustrates the implementation of a multiplier circuit using adders. When multiplying two 3-bit numbers, A2, A1, A0 and B2, B1, B0, the factors are generated and added to obtain the result. As depicted in Fig. 1, the LSB P0, is equivalent to B0A0. Similarly, P1 is the sum of B1A0 and B0A1, P2 is sum of B2A0, B1A1, and B0A2, P3 is sum of B1A2 and B2A1, and the MSB P4, is B2A2.

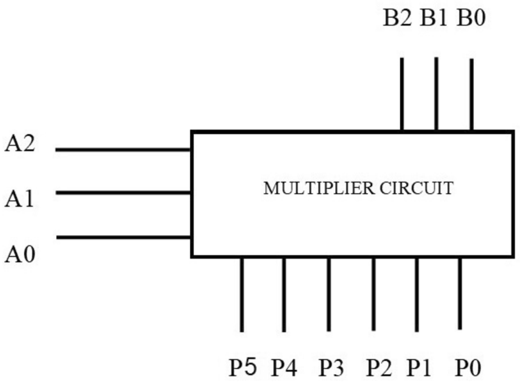


Fig. 2 3 bit multiplier

The implementation of the 3-bit multiplier circuit is shown in Fig. 2 where B2, B1, and B0 and A2, A1, A0 are applied to input pins and the multiplication outcome is displayed on output pins as P4, P3, P2, P1, and P0. Consider the example shown in Fig. 3, where two 3-bit binary numbers 110 and 101 are multiplied, resulting in an output of 11110. Suppose we allow for an error in the LSB P0, since its weight is the smallest and therefore, its error would have the least impact on the output value.

Let's also assume that we allow for errors in P0, P1, and P2, and that these errors cause the output bits P2, P1, and P0 to be reversed, resulting in a value of 001 instead of 110. Therefore, the final output of the multiplier will be 11001 instead of 11110, with a value of 25 instead of 30. The percentage error will be calculated as $((30-25)\div 30)\times 100 = 16.66\%$. The multiplier circuit depicted in Fig. 4a is composed of several blocks, each with an internal circuit illustrated in Fig. 4b,

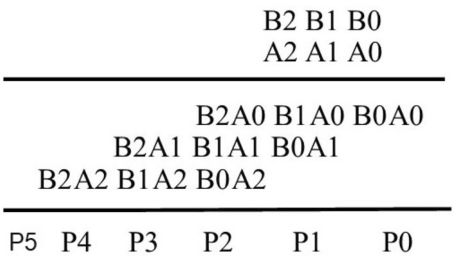


Fig. 1 Multiplication of two 3 bit numbers

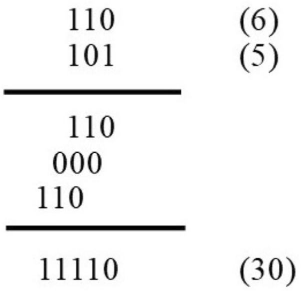
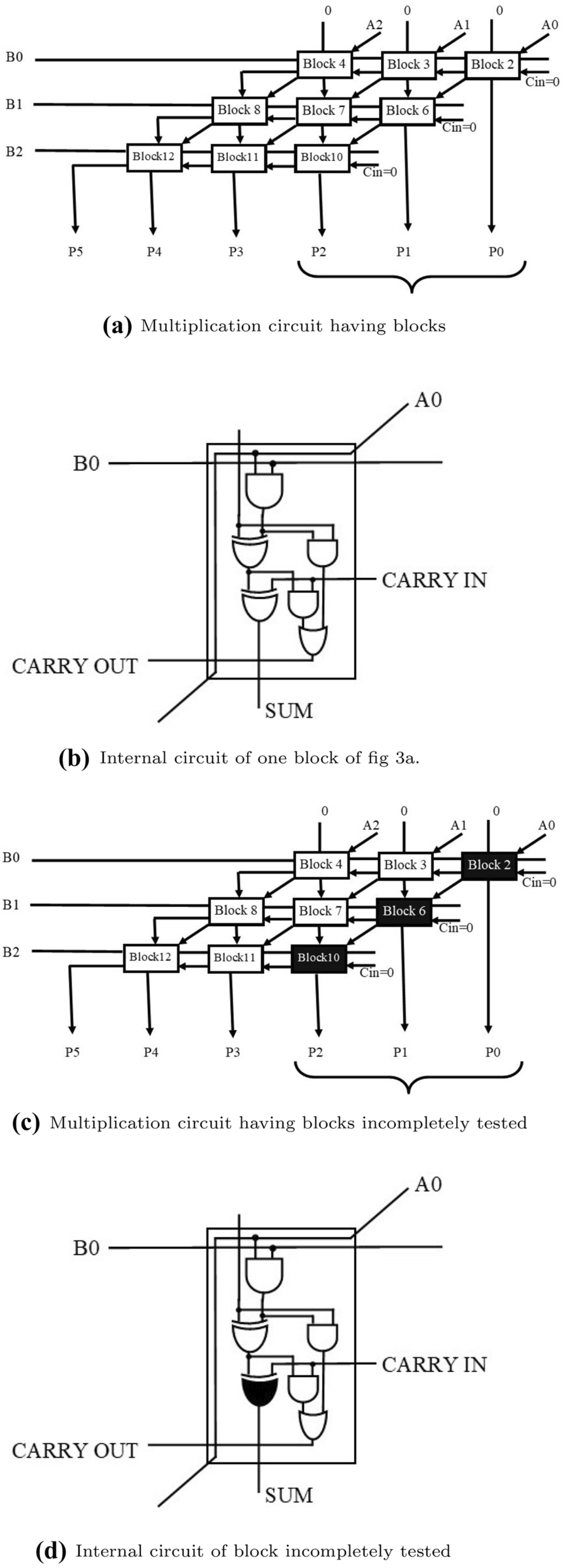


Fig. 3 Example of multiplication of 3 bit numbers



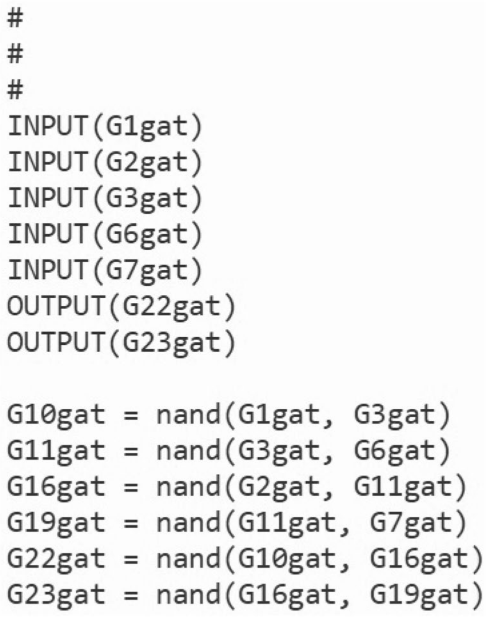
(a) Multiplication circuit having blocks

(b) Internal circuit of one block of fig 3a.

(c) Multiplication circuit having blocks incompletely tested

(d) Internal circuit of block incompletely tested

Fig. 4 Multiplier circuit



```
#
#
#
INPUT(G1gat)
INPUT(G2gat)
INPUT(G3gat)
INPUT(G6gat)
INPUT(G7gat)
OUTPUT(G22gat)
OUTPUT(G23gat)

G10gat = nand(G1gat, G3gat)
G11gat = nand(G3gat, G6gat)
G16gat = nand(G2gat, G11gat)
G19gat = nand(G11gat, G7gat)
G22gat = nand(G10gat, G16gat)
G23gat = nand(G16gat, G19gat)
```

Fig. 5 Benchmark circuit of c17

containing an AND gate and a full adder. The output pins of the multiplier are P5, P4, P3, P2, P1, and P0. To ensure complete testing, all nine blocks corresponding to all output bits need to be tested after implementation, which would require nine units of power if each block takes one unit of power to test. However, if errors in output bits P2, P1, and P0 are tolerable, the corresponding blocks or circuit elements can be left untested, reducing TDV, TAT and TP. As shown in Fig. 4c components in block 2, responsible for output P0, can be left untested if errors on this output pin are tolerable. So, the EX-OR gate inside block 2 shown in Fig. 4d will not be tested. For output P1, blocks 3 and 6 need to be tested but the EX-OR gate inside block 6 can remain untested since it impacts only output P1 and not the other outputs. Similarly, for output P2, blocks 4 and 7 is tested completely, but the EX-OR gate in block 10 responsible for the value on output P1 will not be tested. By this way we can test the SOC incompletely for the least significant outputs.

Outline of the proposed method is given below. Proposed method can be easily understood by the example of circuit c17.

3.1 Step 1:

Design the circuit as shown in Fig. 6 from the benchmark circuit description given in Fig. 5. This circuit consists of five input lines, two output lines and six NAND gates.

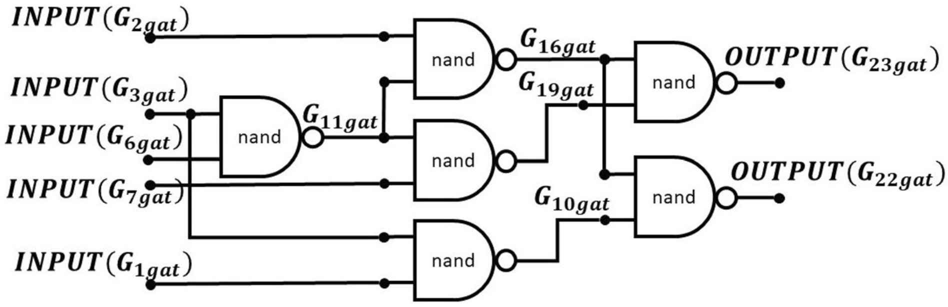


Fig. 6 c17 circuit

3.2 Step 2:

Create a graph that represents the circuit in Fig. 6 by using input pins, output pins, and gates. Five input nodes I_2, I_3, I_6, I_7 and I_1 connected to input lines INPUT(G2gat), INPUT(G3gat), INPUT(G6gat), INPUT(G7gat) and INPUT(G1gat) respectively and two output nodes O_{23} and O_{22} connected to output lines OUTPUT(G23gat) and OUTPUT(G22gat) are created. Six nodes $G_{23}, G_{22}, G_{10}, G_{19}$, G_{16} and G_{11} corresponding to six NAND gates are created.

3.3 Step 3:

Now we have an equivalent graph of the c17 circuit available in Fig. 7.

Our objective is to trace a backtrack path from the fault tolerable output back to the input while ensuring that the gates and connections on this path do not affect other components of the circuit. By doing so, we can guarantee the accuracy of all other outputs except for the output pin O_{23}. Assuming O_{23} can tolerate errors, there is no need to test it. To obtain this backtrack path, we follow Algorithm 1 and Algorithm 2.

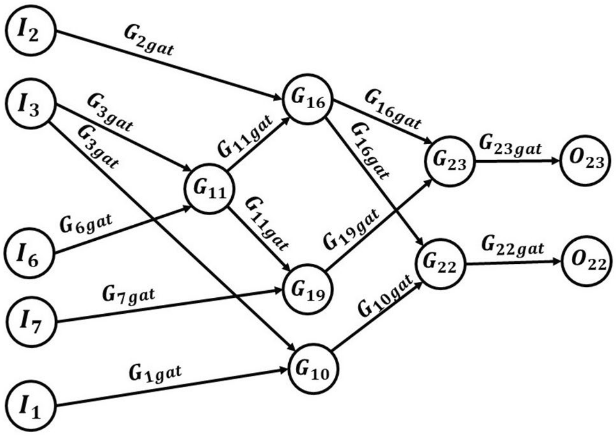


Fig. 7 Equivalent graph of c17 circuit

Algorithm 1 Algorithm for finding GATES not to be tested

```
Require: Circuit graph with N number of nodes,
    M number of output pins not to be tested
Ensure: GATES not to be tested
for node = 1 to N do
    outdegree(node)
end for
for $out\_pin = 1$ to M do
    backtrack ($out\_pin$)
end for
```

Algorithm 2 Backtrack (i)

```
Require: output pin i
Ensure: To find GATES on the path from output
    pin i to one of the input pins
if $outdegree(parents(i)) > 1 \,||\, parents(i) = input$ pin then
    add i to the result
    break
else
    Backtrack (parents(i))
end if
```

Algorithm 1 takes the circuit's equivalent graph as input and outputs a path from the output to the input pin. O_{23} is the *least significant output pin* that can tolerate errors. All the circuit components that O_{23} depends on, need not to be tested if it does not require accuracy on this output pin. Gate G_{23} determines the value on O_{23}, which depends on gates G_{16} and G_{19}. Gate G_{19} remains untested since it does not affect any other component, but G_{16} cannot be left untested as errors in it will impact O_{22}. G_{11} cannot be left untested either, as errors in it will affect G_{16} and, in turn, O_{22}. We can leave input pin I_7 untested since it does not affect any other component

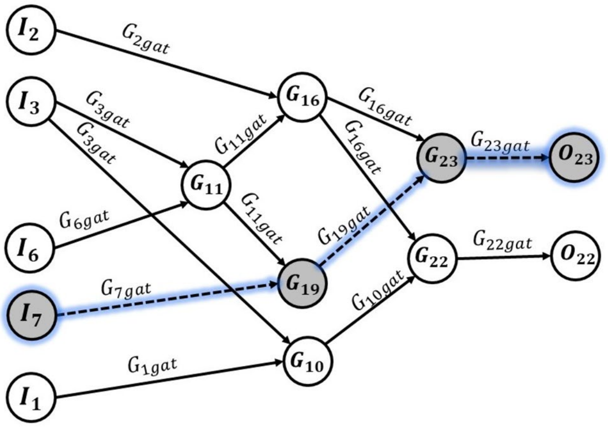


Fig. 8 Backtrack path for c17 circuit

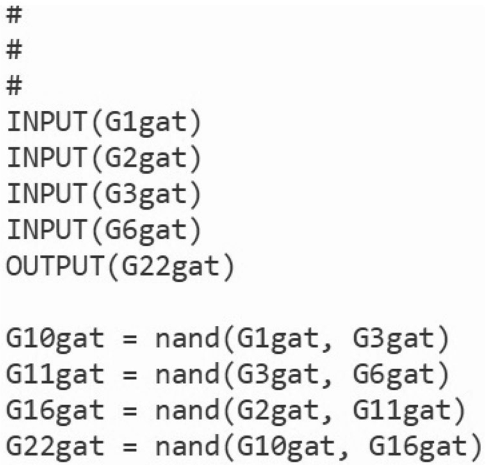
except G_{19}. As shown in Fig. 8, the path obtained using Algorithm 1 is $O_{23} \rightarrow G_{23} \rightarrow G_{19} \rightarrow I_7$, and hardware components on this path will not be tested. This incomplete testing will have impact on testing parameters like FC, TDV, TAT and TP.

4 Impact on Fault Coverage (FC)

After acquiring the path, we eliminate the components along it and conduct a fault simulation.

As seen in Fig. 9, the NAND gates G_{23} and G_{19}, as well as the output pin O_{23} and input pin I_7, all located on the path $O_{23} \rightarrow G_{23} \rightarrow G_{19} \rightarrow I_7$, will not undergo testing. Figure 10 shows the the circuit description of modified circuit for incomplete testing after removing components located on the backtrack path.

In Fig. 11, a list of multiple stuck-at-faults is provided. The total number of faults for complete testing amounts to 22 and 6 of them related to the excluded components, such as $G16gat \rightarrow G23gat/1$, $G23gat/1$, $G23gat/0$, $G11gat \rightarrow G19gat/1$, $G19gat/1$, and $G7gat/1$, will remain untested.



```
#
#
#
INPUT(G1gat)
INPUT(G2gat)
INPUT(G3gat)
INPUT(G6gat)
OUTPUT(G22gat)

G10gat = nand(G1gat, G3gat)
G11gat = nand(G3gat, G6gat)
G16gat = nand(G2gat, G11gat)
G22gat = nand(G10gat, G16gat)
```

Fig. 10 c17 benchmark circuit incomplete testing

Therefore, after excluding these faults, the incomplete testing will account for a total of 16 faults, resulting in a fault coverage reduction of 27.27%. Similarly, for all the cores, a slight compromise with the fault coverage will occur.

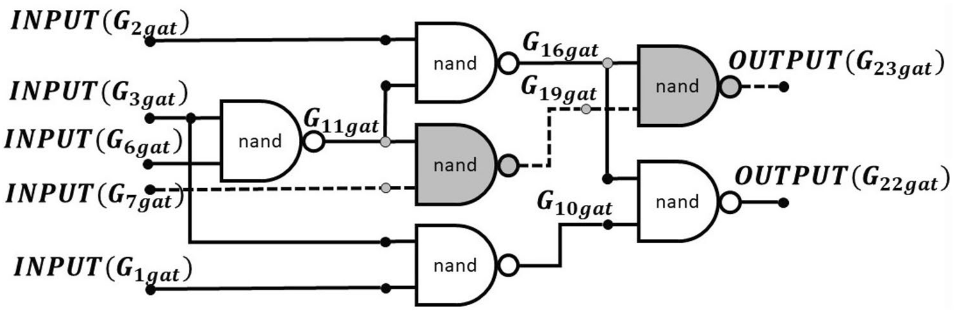
5 Impact on TDV

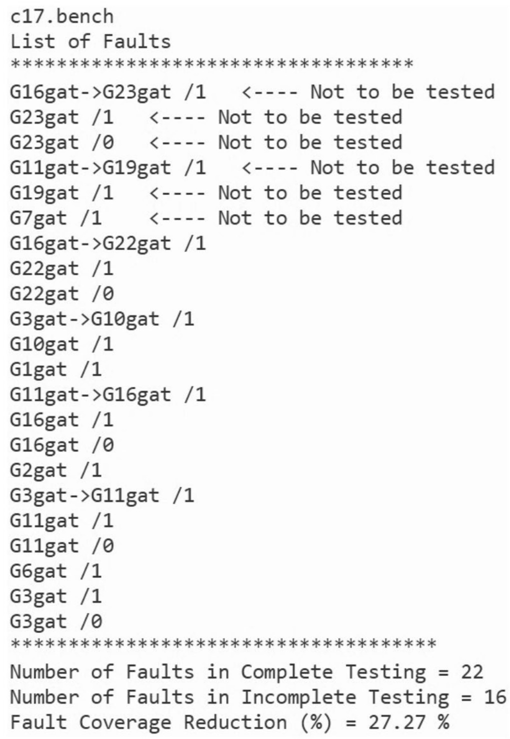
TDV will be reduced in two ways. There are two possible cases.

5.1 Case 1: When Backtrack Path Does Not Terminate on Any Input Node

We can skip testing the circuit elements that lie on the discovered path. By doing so, we reduce the number of required test vectors. For instance, to test the c17 circuit we originally needed seven test patterns to cover all possible faults as shown in Fig. 12. However, by using our incomplete testing method, we can reduce the number of test vectors to five as shown in Fig. 13.

Fig. 9 Equivalent graph of c17 circuit for testing



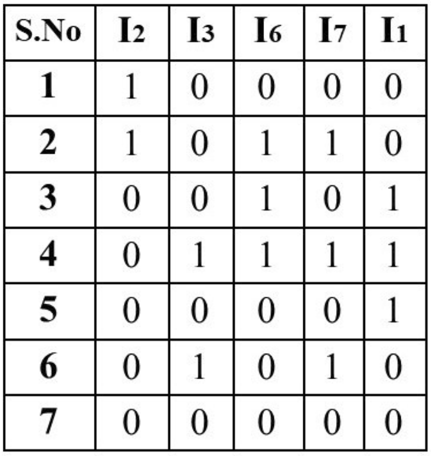


```
c17.bench
List of Faults
***********************************
G16gat->G23gat /1   <---- Not to be tested
G23gat /1   <---- Not to be tested
G23gat /0   <---- Not to be tested
G11gat->G19gat /1   <---- Not to be tested
G19gat /1   <---- Not to be tested
G7gat /1    <---- Not to be tested
G16gat->G22gat /1
G22gat /1
G22gat /0
G3gat->G10gat /1
G10gat /1
G1gat /1
G11gat->G16gat /1
G16gat /1
G16gat /0
G2gat /1
G3gat->G11gat /1
G11gat /1
G11gat /0
G6gat /1
G3gat /1
G3gat /0
*************************************
Number of Faults in Complete Testing = 22
Number of Faults in Incomplete Testing = 16
Fault Coverage Reduction (%) = 27.27 %
```

Fig. 11 c17 benchmark circuit incomplete testing

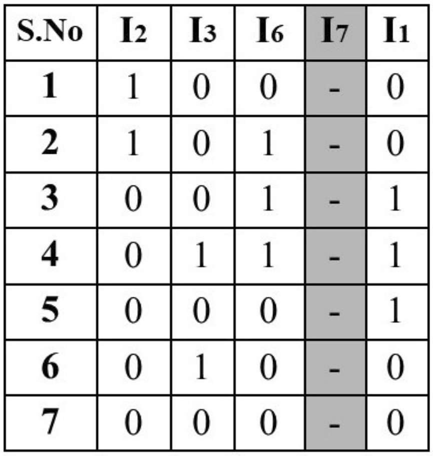
5.2 Case 2: When Backtrack Path Terminates on Any Input Node

The algorithm identifies a path from an output pin to an input pin, indicating that this input pin only affects that specific output pin and not others. Therefore, we can skip testing that output pin by not applying a test bit to its corresponding input pin. This method reduces the number of test vectors and number of bits in each test vector.



S.No	I2	I3	I6	I7	I1
1	1	0	0	0	0
2	1	0	1	1	0
3	0	0	1	0	1
4	0	1	1	1	1
5	0	0	0	0	1
6	0	1	0	1	0
7	0	0	0	0	0

Fig. 12 Test vector set for c17 benchmark circuit

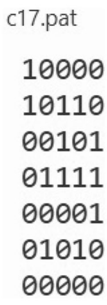


S.No	I2	I3	I6	I7	I1
1	1	0	0	-	0
2	1	0	1	-	0
3	0	0	1	-	1
4	0	1	1	-	1
5	0	0	0	-	1
6	0	1	0	-	0
7	0	0	0	-	0

Fig. 13 Modified test vector set for c17 benchmark circuit

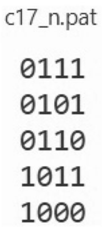
For example, suppose the c17 circuit with five input pins originally required seven test vectors with five bits each, but using our incomplete testing method, we only need five test vectors with four bits per vector. Figure 12 shows the set of test vectors we generated to test the c17 circuit's input pins, including I_1, I_2, I_3, I_6, and I_7. Our reduction in test vectors is achieved by excluding the test bit corresponding to input pin I_7 from all the test vectors located on the backtrack path obtained from Algorithm 1. This is because input pin I_7 is excluded from the circuit during incomplete testing, as shown in Fig. 13. As mentioned earlier, there are two ways to reduce TDV. For instance, in Fig. 14, for complete testing, the c17 circuit has 7 test vectors, each consisting of 5 bits, resulting in a TDV of $7 \times 5 = 35$ bits. On the other hand, for incomplete test shown in Fig. 15 circuit c17 has 5 test vectors, each with 4 bits, resulting in a TDV of $5 \times 4 = 20$ bits. Employing an incomplete test can yield significant TDV savings, which can be calculated as $((35 - 20) \div 35) \times 100 = 42.85\%$.

Fig. 14 Test patterns for c17 benchmark circuit in complete testing



```
c17.pat

10000
10110
00101
01111
00001
01010
00000
```



```
c17_n.pat
0111
0101
0110
1011
1000
```

Fig. 15 Test patterns for c17 benchmark circuit in incomplete testing

6 Impact on TAT

6.1 Computation of TAT

In order to determine the TAT, we have devised an integer Particle Swarm Optimization (PSO) method. This technique was introduced by James Kennedy and Russell Eberhart in 1995 for the optimization of continuous nonlinear functions [21]. Another version of PSO that is designed for binary variables is known as Binary PSO [22]. To calculate the TAT accurately, we must schedule the cores optimally. This involves finding the best possible assignment of cores to the test buses available in the SOC. Suppose an SOC has N_C cores, N_B test buses, and bus widths of $w_1, w_2, ..., w_{N_B}$. In this case, we require an optimal assignment of cores to test buses that results in the minimum TAT. To accomplish this, we propose a PSO-based scheduling method, which is outlined below.

6.1.1 Particle Structure

In our PSO model, the particle is represented by an N_C bit array, with each value being an integer between 1 and N_B. For example, if there are 6 cores and 2 test buses, the particle would be a 6-bit array. Each bit of the array corresponds to a core and has a value of either 1 or 2, indicating whether test bus 1 or test bus 2 is assigned to that core. Thus, the particle structure would be as follows:

2	1	1	1	2	2

This assignment implies that cores 1, 5, and 6 are allocated to test bus 2, while cores 2, 3, 4, 7, and 8 are allocated to test bus 1. We then compute the TAT for each assignment and sum them together. The total sum represents the TAT for the entire SOC, with an optimal distribution of cores to the given buses as represented by the particle.

6.1.2 Initial Population and Position

Population size is 50 for our experiments. All the particles are filled randomly with values between 1 to N_B.

6.1.3 Fitness Function

Suppose a SOC has N_c cores and N_b test buses, with each bus having a width of w_1, w_2, w_3, ... or w_{N_b}. If a core in the SOC has n_i inputs and m_i outputs, and the width of the test bus is greater than the number of core terminals, then we can connect the core pins and test bus in parallel. However, if the width of the test bus is less than the core terminals, then serialization is necessary to apply test vectors [4, 19]. Considering the worst-case scenario where all $w_j - 1$ lines in a bus are connected to $w_j - 1$ core pins in parallel, and the last line in the test bus is connected to the remaining $\psi_i - w_j + 1$ core terminals serially. If core i is assigned to test bus j with a width of w_j, then the TAT T_{ij} in cycles can be calculated using the following equation [3, 18]:

$$T_{ij} = \begin{cases} t_i, & \text{if}(\psi_i \le w_j) \\ t_i \times (\psi_i - w_j + 1), & \text{if}(\psi_i > w_j) \end{cases} \qquad (1)$$

where $\psi_i = MAX(m_i, n_i)$

m_i = number of inputs of core i, $0 < i \le N_c$
n_i = number of outputs of core i, $0 < i \le N_c$

$$t_i = \begin{cases} p_i, & \text{for combinational core} \\ (p_i + 1) \times f_i / N_i + p_i, & \text{for cores with internal scan chain} \end{cases} \qquad (2)$$

where core i contains f_i flip-flops and N_i internal scan chains.

p_i is number of test patterns.

TAT for SOC is summation of TAT for each core where each core is optimally allocated to test bus.

$$\mathcal{T} = \max_j \sum_{i=1}^{N_c} T_{ij}, \text{ where } 1 \le j \le N_b \qquad (3)$$

Fitness function of our PSO based scheduling algorithm is to *minimize* $\mathcal{T}$.

6.2 Reduction in TAT

TAT will be reduced in two ways in incomplete testing.

6.2.1 When Backtrack Path Does Not Terminate on Any Input Node

Suppose p_r number of test pattern removed from test vector set in incomplete testing then reduced TAT will calculated as follows:

$$T_{ij} = \begin{cases} t_i, & \text{if}(\psi_i \le w_j) \\ t_i \times (\psi_i - w_j + 1), & \text{if}(\psi_i > w_j) \end{cases} \qquad (4)$$

where $\psi_i = MAX(m_i, n_i)$

m_i = number of inputs of core i, $0 < i \leq N_c$
n_i = number of outputs of core i, $0 < i \leq N_c$

$$t_i = \begin{cases} p_i - p_r, & \text{for combinational core} \\ ((p_i - p_r) + 1) \times f_i / N_i + (p_i - p_r), & \\ & \text{for cores with internal scan chain} \end{cases} \quad (5)$$

where core i contains f_i flip-flops and N_i internal scan chains. p_i is the number of test patterns.

6.2.2 When Backtrack Path Terminates on Any Input Node

The Algorithm 1 identifies a path from an output pin to an input pin, indicating that this input pin only affects that specific output pin and not others. Therefore, we can skip testing that output pin by not applying a test bit to its corresponding input pin. Suppose there are L number of bit positions where test bits will be removed from all the test vector then

$$\psi_i = \begin{cases} m_i - L, & \text{if}(m_i > n_i) \\ n_i, & \text{if}(m_i < n_i) \end{cases} \quad (6)$$

After removing bits from the L designated bit positions in all test vectors, reduced TAT will be calculated as follows :

$$T_{ij} = \begin{cases} t_i, & \text{if}(\psi_i \leq w_j) \\ t_i \times ((m_i - L) - w_j + 1), & \\ & \text{if}(\psi_i > w_j) and (m_i > n_i) \\ t_i \times (n_i - w_j + 1), & \\ & \text{if}(\psi_i > w_j) and (m_i < n_i) \end{cases} \quad (7)$$

Net reduction in TAT after incomplete testing will be calculated as follows:

$$S_{ij} = \begin{cases} Zero, & \text{if}(\psi_i \leq w_j) \\ t_i \times ((L) - w_j + 1), & \\ & \text{if}(\psi_i > w_j) and (m_i > n_i) \\ Zero, & \\ & \text{if}(\psi_i > w_j) and (m_i < n_i) \end{cases} \quad (8)$$

7 Impact on TP

7.1 Computation of TP

The TP can be calculated using a metric derived by calculating the switching activities in the test vector bits , which refer to transitions from 1 to 0 or from 0 to 1 [34]. Test vectors can be applied to the cores in two ways:

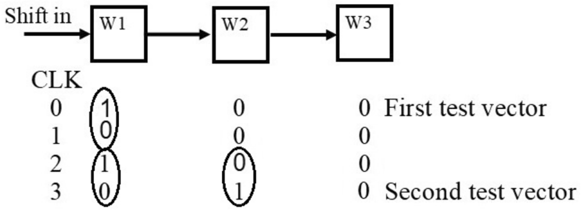


Fig. 16 Transition during shift in of first and second test vector [34]

7.1.1 Parallel Application

When test bits are applied to the core in parallel, transitions can be calculated using the *hamming distance* between two test vectors. For example, if we have two vectors, 0 1 1 1 1 and 1 1 1 1 0, and they are applied in parallel one after the other, the total number of transitions from 0 to 1 or 1 to 0 would be two.

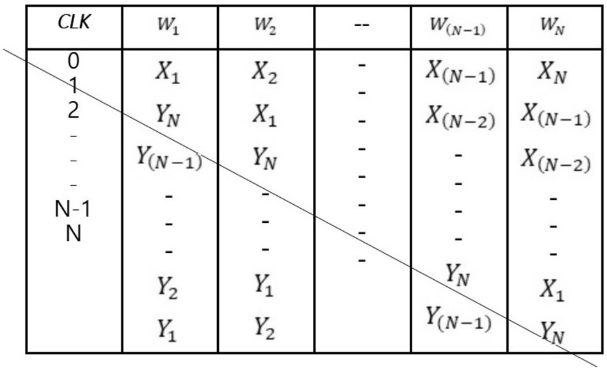
7.1.2 Serial Application

Alternatively, test vectors can be applied sequentially to the wrapper chain, as shown in Fig. 16. In Fig. 16 two test vectors, 100 and 010 have been applied serially to the wrapper chain. The total number of transitions is three in this way.

Figure 17 presents a general illustration of serially shifting two test vectors, namely $X_1, X_2, X_3, ..., X_n$ and $Y_1, Y_2, Y_3, ..., Y_n$. In Eq. (9a), the term t_{11} represents the transition in the first wrapper cell, as shown in Fig. 17. On the other hand, the term t_{1j} in Eq. (9b) is obtained from the upper triangle of the matrix illustrated in Fig. 17, while the term t_{i1} in Eq. (9d) is derived from the lower triangle of the matrix depicted in Fig. 18.

$$t_{11} = X_1 \oplus Y_N \quad (9a)$$

$$t_{1j} = X_{j-1} \oplus X_j \qquad (1 < j \leq N) \quad (9b)$$



CLK	W_1	W_2	--	$W_{(N-1)}$	W_N
0 1 2 - - - N-1 N	X_1 Y_N $Y_{(N-1)}$ - - - Y_2 Y_1	X_2 X_1 Y_N - - - Y_1 Y_2	- - - - - - - -	$X_{(N-1)}$ $X_{(N-2)}$ - - - - Y_N $Y_{(N-1)}$	X_N $X_{(N-1)}$ $X_{(N-2)}$ - - - X_1 Y_N

Fig. 17 Transition matrix [34]

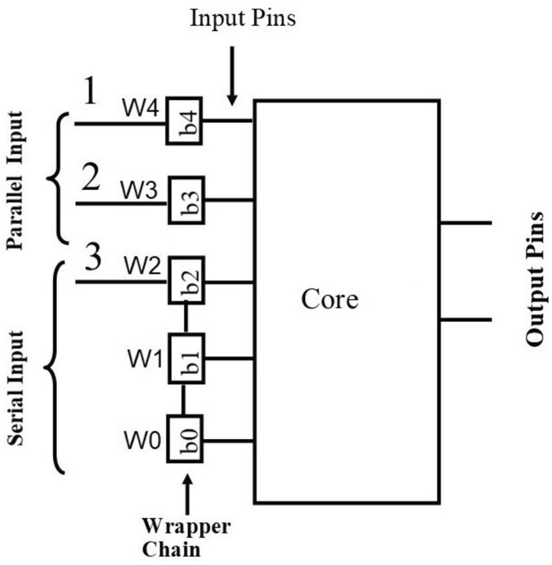


Fig. 18 TAM architecture for complete testing

$$t_{i1} = Y_{N-i+2} \oplus Y_{N-i+1} \qquad (1 < i \le N) \tag{9c}$$

$$t_{ij} = t_{(i-1)(j-1)} \qquad (1 < i,j \le N) \tag{9d}$$

To get the number of transitions a $n \times n$ *transition matrix* $T(X, Y)$ defined in Eq. (10) [34]

$$T(X, Y) = \begin{pmatrix} t_{11} & \dots & t_{1n} \\ \vdots & & \vdots \\ t_{n1} & \dots & t_{nn} \end{pmatrix} \tag{10}$$

where t_{ij}= 1, if transition occurs in j^{th} wrapper cell in i^{th} clock cycle, otherwise $t_{ij} = 0$.

To determine the total number of transitions in the sequential application of test vectors to the wrapper chain, Eq. (11) is utilized, whereby the summation of all transition components in the transition matrix is performed. In this equation, tr_{total} represents the total number of transitions.

Total Number of Transitions (tr_{total}) [34]:

$$tr_{total} = \sum_{i=1}^{n} \sum_{j=1}^{n} t_{ij} \tag{11}$$

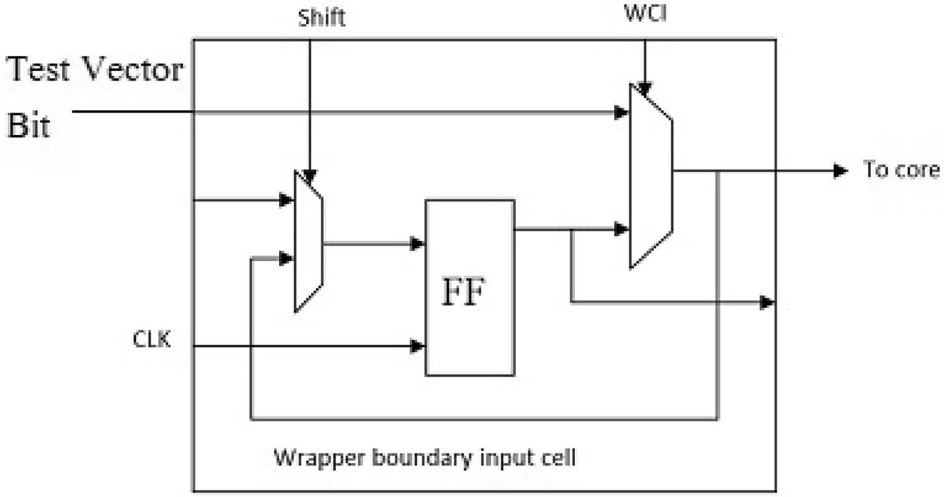
7.2 TAM Architecture for Incomplete Testing

A test architecture design for applying a test vector to core is shown in Fig. 18.

The test vectors are applied to the core using a wrapper chain, which consists of a series of wrapper boundary cells (WBCs) connected together. Test vectors can be applied to the input pins through the connected WBC.

The internal structure of the WBC is depicted in Fig. 19. A WBC can insert a test bit to a core or can be transparent to the test bit. When WBC is transparent to the test bit then it simply transfers that test bit to the next WBC in the wrapper chain instead of inserting it into the core input pin. The core has five input pins, the wrapper chain associated with these input pins comprises five WBCs, namely W_4, W_3, W_2, W_1, and W_0. and test bus of width three is considered in Fig. 18. Test bus width is three means that there are three wires in test bus namely 1, 2 and 3. Two out of these three test wires 1 and 2 are connected to WBC W_4 and W_3 respectively while bus wire 3 is connected to WBC W_2 W_2 and W_1 serially. So if we shift any test vector bit to W_2 then it can be shifted serially to W_1 or W_0 (Fig. 20).

Fig. 19 Wrapper Boundary Cell



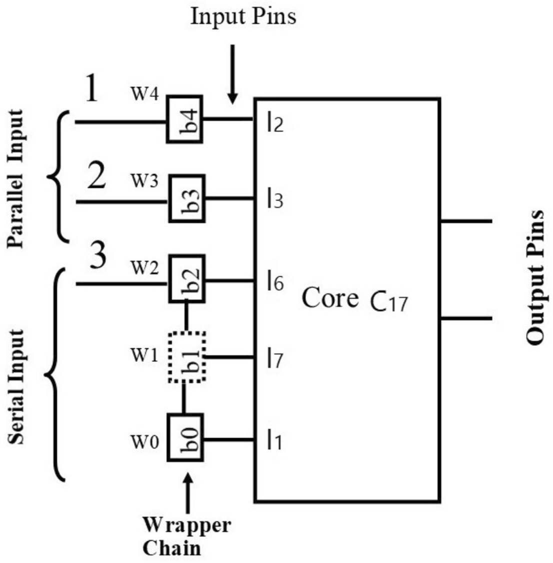


Fig. 20 Modified TAM architecture for incomplete testing

7.3 Reduction in TP in Incomplete Testing

Consider the example depicted in Fig. 12, where input pins I_2, I_3, I_6, I_7, and I_1 of core c17 are fed with test vectors b_4, b_3, b_2, b_1, and b_0 respectively. Test vector bits b_4 and b_3 are transported through test bus wires 1 and 2 and applied to input pins I_2 and I_3 via WBC W_4 and W_3 in parallel, while test vector bits b_2, b_1, and b_0 are transported on test bus wire 3 and applied to input pins I_6, I_1, and I_7 through WBCs W_2, W_1, and W_0 sequentially. As a result, when test vectors are applied one after another, the transitions in test bit values will be calculated in two ways: the transition in W_4 and W_3 will be calculated by hamming distance due to parallel insertion, and the transition in WBCs W_2, W_1, and W_0 will be calculated by Eq. (11) due to serial insertion. This will result in a reduction of TP consumption in two ways. Firstly, in incomplete testing, a smaller number of test vectors will lead to fewer bit flips, ultimately reducing TP. Secondly, in incomplete testing, pin I_7 will not be tested, and test WBC W_1 will become transparent to input bits. Consequently, all the bit flips in WBC W_1 corresponding to input pin I_7 will be eliminated straightforwardly. Fewer transitions will reduce the TP.

8 Experimental Results

To conduct the experiments, the SOC was utilized, and the ISCAS'85 and ISCAS'89 benchmark circuits were employed for testing purposes [42]. The experiment program was coded in C++, while the test patterns were generated using ATALANTA [43]. Additionally, the HOPE simulator was utilized for fault simulation [44]. All the experiments were carried out on the SOC shown in Fig. 21, which consists of eight cores (c880, c2670, c432, c6288, s27, s298, s444, s520) and two test buses. The total bus width of 48 is distributed between both buses.

8.1 Results for FC and TDV

The results for FC and TDV are presented in Table 1. Test patterns and fault lists were generated for all the cores. For complete testing a backtracking approach was used to exclude hardware components associated with the backtrack path from testing explained in section 4. Faults corresponding to excluded hardwares is also removed. The total number of faults for complete and incomplete testing are displayed in columns 2 and 4 of Table 1, respectively, while the fault reduction is shown in column 6. The TDV can be calculated as the product of the number of test vectors and the number of bits in a test vector. The total TDV for complete and incomplete testing are shown in columns 3 and 5, respectively, while the reduction in TDV is shown in column 7. The TDV comparison between complete and incomplete testing is provided in Table 1. For cores c880, c2670, c432, c6288, s27, s298, s444, and s520 the TDV reduction is 27.78%, 45.87%, 52.17%, 40.96%, 33.33%, 39.29%, 43.33% and 48.77% respectively, while the fault coverage is compromised up to 0.76%, 3.50%, 1.89%, 0.87%, 18.75%, 12.01%, 0.84% and 15.14% respectively.

8.2 Results for TAT

Table 2 presents the TAT for the optimal allocation of cores to the test buses in the SOC. We used a PSO-based scheduling algorithm proposed in section 6.1 for the optimal allocation. The test bus has a width of 48, and we conducted experiments for various bus width distributions, such as

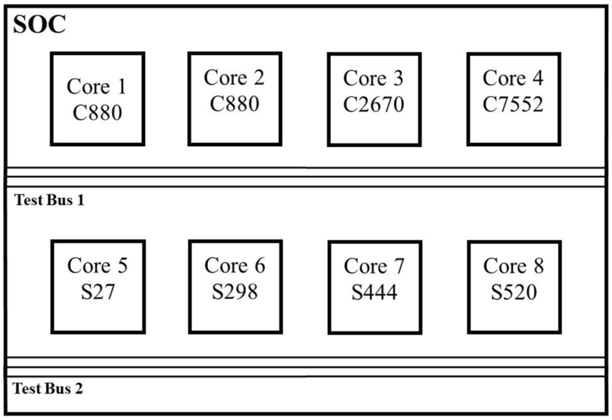


Fig. 21 SOC used for experiments

Table 1 Results for FC and TDV

Cores	No of faults in complete testing	TDV for complete testing	No. of faults in incomplete testing	TDV for incomplete testing	Fault reduction (%)	TDV reduction (%)
c432	524	2592	520	1872	0.76	27.78
c880	942	6000	909	3248	3.50	45.87
c2670	2747	45435	2695	21733	1.89	52.17
c7552	7550	74313	7484	43878	0.87	40.96
s27	32	63	26	42	18.75	33.33
s298	308	952	271	578	12.01	39.29
s444	474	1320	470	748	0.84	43.33
s520	555	2688	471	1377	15.14	48.77

(1,47), (4,44), (28,20), (24,24), (32,16), and (40,8), which are shown in the first column of Table 3. The optimal core-to-bus allocation is depicted in columns 2 and 4, and the TAT for complete and incomplete testing is represented in columns 3 and 5, respectively. The reduction in TAT is shown in column 6. For example, in row 1 of Table 2, the bus width distribution between two test buses is (1,47), and the optimal core-to-test bus allocation is 11122121, which indicates that test bus 1 is allocated to core 1, 2, 3, 6, and 8, while test bus 2 is allocated to core 4, 5, and 7. The TAT for this optimal distribution of 11122121 is minimized TAT for SOC, which is 57863 cycles for complete testing and 30074 cycles for incomplete testing. The TAT can be calculated in real-time according to the system's frequency. The reduction in TAT for this optimal distribution is shown in row 1, column 6, which is 48.03% for SOC. Similarly, TAT is calculated for other combinations of test bus width distributions, and for each distribution, the savings are around 48% on average, which is significant.

8.3 Results for TP

8.3.1 TP for Complete Testing

Table 3 shows the TP results obtained from experiments conducted to calculate the TP for complete testing using different random bus width distributions and optimal core-to-bus scheduling, as described in section 7.1. In column 2, the TP is calculated for all eight cores and the SOC, where the bus width distribution is (1,47), and the core-to-bus allocation is 11122121. The TP calculated for cores c880, c2670, c432, c6288, s27, s298, s444, and s520 are 44779, 187377, 5090603, 7475138, 223, 10303, 651, and 38519, respectively. The TP for complete testing of the SOC is 12847593, 11178947, 11373648, 11127686, 12180563, and 12222238 for bus width distributions of (1,47), (4,44), (28,20), (24,24), (32,16), and (40,8), respectively.

8.3.2 TP for Incomplete Testing

The results of experiments for calculating TP for incomplete testing with various bus width distributions and optimal core-to-bus scheduling are presented in Table 4. The calculations were performed using the method described in section 7.1. In column 2 of the table, the TP for all eight cores and SOC is shown for the bus width distribution of (1, 47) and core to bus allocation of 11122121. The TP values for individual cores such as c880, c2670, c432, c6288, s27, s298, s444, and s520 are 31978, 97835, 2154776, 4302353, 112, 6549, 346, and 22443 respectively. The TP for incomplete testing of SOC is 6616392, 5858528, 5746942, 5602130, 6316776, and 6241246 for different bus width distributions, namely, (1, 47), (4, 44), (28, 20), (24, 24), (32, 16), and (40, 8) respectively.

Table 2 Results for TAT

Buswidth distribution	Optimal allocation	TAT for SOC (complete test)	Optimal allocation	TAT for SOC (incomplete test)	Reduction in TAT (%)
(1, 47)	11122121	57863	11122121	30074	48.03
(4, 44)	11122111	58885	11122111	30679	47.90
(28, 20)	22212222	64620	22212222	34080	47.26
(24, 24)	22212222	66056	22212222	34932	47.12
(32, 16)	22212222	63184	22212222	33228	47.41
(40, 8)	22212222	60312	22212222	31524	47.73

Table 3 Results for TP for complete testing

Bus width distribution	(1, 47)	(4, 44)	(28, 20)	(24, 24)	(32, 16)	(40, 8)
Optimal allocation	11122121	11122111	22212222	22212222	22212222	22212222
Cores	TP	TP	TP	TP	TP	TP
c432	44779	37605	50050	50050	1945	50050
c880	187377	187377	55698	69008	187377	138426
c2670	5090603	3417327	5178381	5178381	4464549	5178381
c7552	7475138	7475138	6068577	5809305	7475138	6831883
s27	223	73	341	341	33	341
s298	10303	10303	463	463	10303	3019
s444	651	12605	18881	18881	2699	18881
s526	38519	38519	1257	1257	38519	1257
SOC	12847593	11178947	11373648	11127686	12180563	12222238

Table 4 Results for TP for incomplete testing

Bus width distribution	(1, 47)	(4, 44)	(28, 20)	(24, 24)	(32, 16)	(40, 8)
Optimal allocation	11122121	11122111	22212222	22212222	22212222	22212222
Cores	TP	TP	TP	TP	TP	TP
c432.pat	31978	27402	36088	36088	1456	36088
c880.pat	97835	97835	28209	35299	97835	71851
c2670.pat	2154776	1395630	2196792	2196792	1884964	2196792
c7552.pat	4302353	4302353	3474765	3322563	4302353	3923667
s27.pat	112	30	199	199	16	199
s298.pat	6549	6549	277	277	6549	2037
s444.pat	346	6286	9949	9949	1160	9949
s526.pat	22443	22443	663	963	22443	663
SOC	6616392	5858528	5746942	5602130	6316776	6241246

Table 5 Results for comparison of TP

Bus width Distribution	Optimal allocation	TP for SOC (complete test)	Optimal allocation	TP for SOC (incomplete test)	Reduction in TP (%)
(1, 47)	11122121	12847593	11122121	6616392	48.50
(4, 44)	11122111	11178947	11122111	5858528	47.59
(28, 20)	22212222	11373648	22212222	5746942	49.47
(24, 24)	22212222	11127686	22212222	5602130	49.66
(32, 16)	22212222	12180563	22212222	6316776	48.14
(40, 8)	22212222	12222238	22212222	6241246	48.94

8.3.3 Comparison of TP for Complete and Incomplete Testing

The TP results for both complete and incomplete testing are presented in Table 5. Column 1 and 2 display the bus width distribution and optimal core-to-bus allocation, respectively. Column 3 and 5 show the TP of SOC for complete and incomplete testing, respectively. Finally, column 6 displays the reduction in TP. For instance, row 1 of Table 5 demonstrates the bus width distribution of (1,47), with the optimal core-to-bus allocation of 11122121. The TP for this optimal distribution is 12847593 for complete testing and 6616392 for incomplete testing. Row 1, column 6 of Table 5 indicates a TP reduction of 48.50% for SOC. Similarly, TP is calculated for other test bus width distribution combinations. For the bus width distribution of (24,24), the saving is up to 49.66%. On average, the saving for other distributions is also around 49%, which is highly significant.

9 Conclusion

As per our knowledge, proposed method is novel and beneficial in various fault-tolerant applications, including audio, video, graphics, wireless communications, multilevel cell STT-RAM, and neural networks. In such cases, if some output bits can tolerate errors, computation can be faster. Nevertheless, this strategy can be particularly useful for large SOC devices where extensive testing may not be practical due to TAT and TP constraints. Incomplete testing is a useful tool for optimizing testing parameters such as TDV, TAT, and TP, with a minor compromise in FC. This approach is not only faster but also cost-effective, which can increase customer access to devices. It. Our research shows that incomplete testing can reduce TDV, TAT, and TP by up to 52%, 48%, and 50%, respectively, with a trade-off of 1% to 10% in FC.

10 Future Work

The current paper focuses only on stuck-at-faults, but in future research, we can explore other types of faults such as bridge faults, delay faults etc. Furthermore, our method can be extended to 3D hierarchical SOCs. Additionally, we can propose an incomplete testing approach for specific applications such as image processing and more generally, digital signal processing, wireless communication and other error tolerant applications.

Data Availability The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request. All data generated or analysed during this study are included in this published article.

Declarations

Competing Interests The authors have no relevant financial or non-financial interests to disclose. The authors have no competing interests to declare that are relevant to the content of this article. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript. The authors have no financial or proprietary interests in any material discussed in this article.

References

1. Alampally S, Venkatesh RT, Shanmugasundaram P, Parekhji RA, Agrawal VD (2011) An efficient test data reduction technique through dynamic pattern mixing across multiple fault models. In 29th VLSI Test Symposium, pp 285–290
2. Biswas SN, Das SR, Petriu EM (2014) On system-on-chip testing using hybrid test vector compression. IEEE Trans Instrument Measure 63(11):2611–2619
3. Chakrabarty K (2000) Design of system-on-a-chip test access architectures using integer linear programming. In Proceedings 18th IEEE VLSI Test Symposium, IEEE, pp 127–134
4. Chakrabarty K (2001) Optimal test access architectures for system-on-a-chip. ACM Trans Des Autom Electron Syst (TODAES) 6(1):26–49
5. Chandra A, Chakrabarty K (2000) Test data compression for system-on-a-chip using Golomb codes. In Proceedings of 18th IEEE VLSI Test Symposium, pp 113–120
6. Chandra A, Chakrabarty K (2001) Combining low-power scan testing and test data compression for system-on-a-chip. In Proceedings of the 38th Design Automation Conference (IEEE Cat. No. 01CH37232), pp 166–169
7. Chandra A, Chakrabarty K (2001) System-on-a-chip test-data compression and decompression architectures based on Golomb codes. IEEE Trans Comput Aided Des Integr Circ Syst 20(3):355–368
8. Chandra A, Chakrabarty K (2002) Reduction of SOC test data volume, scan power and testing time using alternating run-length codes. In Proceedings of 2002 Design Automation Conference (IEEE Cat. No. 02CH37324), pp 673–678
9. Chandra A, Chakrabarty K (2002) Test data compression and decompression based on internal scan chains and Golomb coding. IEEE Trans Comput Aided Des Integr Circ Syst 21(6):715–722
10. Chandrasekharan A, Eggersglüß S, Große D, Drechsler R (2018) Approximation-aware testing for approximate circuits. In 2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC), pp 239–244
11. Chattopadhyay S, Reddy KS (2003) Genetic algorithm based test scheduling and test access mechanism design for system-on-chips. In Proceedings of 16th International Conference on VLSI Design, 2003. pp 341–346
12. Djahromi A, Eltawil A, Kurdahi F (2007) Exploiting fault tolerance towards power efficient wireless multimedia applications. In 2007 4th IEEE Consumer Communications And Networking Conference, pp 400–404
13. Dutt S, Chauhan A, Bhadoriya R, Nandi S, Trivedi G (2015) A high-performance energy-efficient hybrid redundant mac for error-resilient applications. In 2015 28th International Conference on VLSI Design. IEEE, pp 351–356
14. Eltawil A, Kurdahi F (2005) Improving effective yield through error tolerant system design. In 2005 12th IEEE International Conference on Electronics, Circuits and Systems, pp 1–4
15. Gebregiorgis A, Tahoori M (2019) Test pattern generation for approximate circuits based on Boolean satisfiability. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1028–1033
16. Giri C, Tipparthi DKR, Chattopadhyay S (2007) Genetic algorithm based approach for hierarchical SOC test scheduling. In 2007 International Conference on Computing: Theory and Applications (ICCTA'07), pp 141–145
17. Giri C, Sarkar S, Chattopadhyay S (2007) A genetic algorithm based heuristic technique for power constrained test scheduling in core-based SOCs. In 2007 IFIP International Conference on Very Large Scale Integration, pp 320–323
18. Harmanani HM, Sawan R (2007) Test bus assignment, sizing, and partitioning for system-on-chip. Can J Electr Comput Eng 32(3):165–175
19. Iyengar V, Chakrabarty K, Marinissen EJ (2002) Test wrapper and test access mechanism co-optimization for system-on-chip. J Electron Test 18(2):213–230
20. Jas A, Ghosh-Dastidar J, Ng ME, Touba NA (2003) An efficient test vector compression scheme using selective Huffman coding. IEEE Trans Comput Aided Des Integr Circ Syst 22(6):797–806
21. Kennedy J, Eberhart R (1995) Particle swarm optimization. In Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp 1942–1948
22. Kennedy J, Eberhart RC (1997) A discrete binary version of the particle swarm algorithm. In 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, vol. 5, pp 4104–4108
23. Li J, Hu Y, Li X (2007) Impact-factor-guided x-filling for peak power reduction during test. In TENCON 2007 - 2007 IEEE Region 10 Conference, pp 1–4

24. Li L, Chakrabarty K, Touba N (2003) Test data compression using dictionaries with selective entries and fixed-length indices. ACM Trans Des Autom Electr Syst 8:470–490
25. Lin X, Rajski J, Pomeranz I, Reddy SM (2001) On static test compaction and test pattern ordering for scan designs. In Proceedings of International Test Conference 2001 (Cat. No. 01CH37260), pp 1088–1097
26. Liu W, Lombardi F, Shulte MA (2020) Retrospective and prospective view of approximate computing. Point of View. Proceedings of the IEEE
27. Lu S, Chuang H, Lai G, Lai B, Huang Y (2009) Efficient test pattern compression techniques based on complementary Huffman coding. In 2009 IEEE Circuits and Systems International Conference on Testing and Diagnosis, pp 1–4
28. Mehla US, Dasgupta KS, Devashrayee NM (2010) Hamming distance based reordering and column wise bit stuffing with difference vector: a better scheme for test data compression with run length based codes. In 2010 23rd International Conference on VLSI Design, pp 33–38
29. Mitra S, Kim KS (2002) X-compact: an efficient response compaction technique for test cost reduction. In Proceedings of International Test Conference, pp 311–320
30. Mitra S, Kim KS (2004) X-compact: an efficient response compaction technique. IEEE Trans Comput Aided Des Integr Circ Syst 23(3):421–432
31. Moons B, De Brabandere B, Van Gool L, Verhelst M (2016) Energy-efficient convnets through approximate computing. In 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), pp 1–8
32. Rao V, Nowrouxian B (1996) A novel high-speed parallel multiply-accumulate arithmetic architecture employing modified radix-4 signed-binary recoding. In Proceeding of the 39th Midwest Symposium on Circuits and Systems, IEEE, vol. 1, pp 57–60
33. Rosinger PM, Al-Hashimi BM, Nicolici N (2002) Power profile manipulation: a new approach for reducing test application time under power constraints. IEEE Trans Comput Aided Des Integr Circ Syst 21(10):1217–1225
34. Samii S, Larsson E, Chakrabarty K, Peng Z (2006) Cycle-accurate test power modeling and its application to SOC test scheduling. In Test Conference, ITC'06. IEEE International. IEEE, pp 1–10
35. Sampaio F, Shafique M, Zatt B, Bampi S, Henkel J (2015) Approximation-aware multi-level cells STT-RAM cache architecture. In 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), pp 79–88
36. Sivanantham S, Manuel JP, Sarathkumar K, Mallick PS, Perinbam JRP (2012) Reduction of test power and test data volume by power aware compression scheme. In 2012 International Conference on Advances in Computing and Communications, pp 158–161
37. Shin D, Gupta S (2010) Approximate logic synthesis for error tolerant applications. In 2010 Design, Automation & Test In Europe Conference & Exhibition (DATE 2010), pp 957–960
38. Torres-Huitzil C, Girau B (2017) Fault and error tolerance in neural networks: a review. IEEE Access 5:17322–17341
39. Traiola M, Virazel A, Girard P, Barbareschi M, Bosio A (2018) Testing approximate digital circuits: Challenges and opportunities. In 2018 IEEE 19th Latin-American Test Symposium (LATS), pp 1–6
40. Wu T-B, Liu H-Z, Liu P-X (2013) Efficient test compression technique for SOC based on block merging and eight coding. J Electron Test 29(6):849–859. [Online]. Available at: https://doi.org/10.1007/s10836-013-5415-7
41. Yu Y, Yang Z, Peng X (2012) Test data compression based on variable prefix dual-run-length code. In 2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings, pp 2537–2542
42. [Online]. Available: http://ddd.fit.cvut.cz/prj/Benchmarks/. Accessed Feb 2019
43. [Online]. Available: http://ddd.fit.cvut.cz/prj/Atalanta-M/. Accessed Feb 2019
44. [Online]. Available: http://ddd.fit.cvut.cz/index.php?page=download. Accessed Feb 2019

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Kunwer Mrityunjay Singh is a researcher and lecturer hailing from India. He earned his Bachelor of Science degree in Physics, Chemistry, and Mathematics from Ewing Christian College in Allahabad, Uttar Pradesh in 2007. Subsequently, he pursued a Bachelor of Technology in Electronics and Communication Engineering from the University of Allahabad in 2010. After completing his undergraduate studies, he embarked on a dual degree program of M.Tech. and Ph.D. in the Department of Computer Science at the Indian Institute of Technology in Guwahati, India. He currently works as a lecturer of Electronics Engineering in the Department of Technical Education, Government of Uttar Pradesh, having secured the second rank statewide in a highly competitive competition organized by the Uttar Pradesh Public Service Commission (UPPSC) to obtain the position. He has over six years of teaching experience and is a qualified Graduate Aptitude Test in Engineering (GATE) candidate, having successfully cleared the test in 2011, 2012, and 2013. His research interests include VLSI testing and design for testability, fault tolerance, network on chip, and embedded systems, and he has published research papers in SOC testing and NOC testing.

Jatindra Kumar Deka is an academic and researcher who received his B.E. degree in Electronics from the Motilal Nehru National Institute of Technology Allahabad, India in 1988. He went on to pursue his M.Tech in Computer Science and Information Technology from the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India, in 1993. Later, in 2001, he obtained his Ph.D. from the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur. Currently, Jatindra is a Professor in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, India, where he is engaged in academic and industry-sponsored research related to VLSI Testing and Design for Testability. His research interests include Formal Modeling and Verification, CAD for VLSI and Embedded Systems (Design, Testing, and Verification), and Data Mining. He has published more than 40 research papers to his credit and is a member of the Institute of Electrical and Electronics Engineers (IEEE).

Santosh Biswas is a researcher and academician from India. He received his B.E. degree in 2001 from the National Institute of Technology Durgapur, India. Subsequently, he pursued his MS from the Department of Electrical Engineering at the Indian Institute of Technology Kharagpur, India, and graduated with the highest institute CGPA in 2004. In 2008, he completed his PhD from the Department of Computer Science and Engineering at the Indian Institute of Technology Kharagpur. Afterward, he joined the Department of Computer Science and Engineering at the Indian Institute of Technology Guwahati in 2009 and is currently serving as an Associate Professor. Dr. Biswas has been an integral part of several research projects sponsored by Industry and Government agencies. His research interests include VLSI

Testing and Design for Testability, Fault Tolerance, Network Security, Discrete-event systems, and Embedded Systems. He has published more than 130 research papers and is a member of IEEE. Along with academic research, he is also actively engaged in industry-sponsored research in VLSI Testing and Design for Testability. Dr. Biswas is highly regarded for his contributions to the field and his expertise in VLSI testing and fault-tolerant computing.